B Trees

Motivation

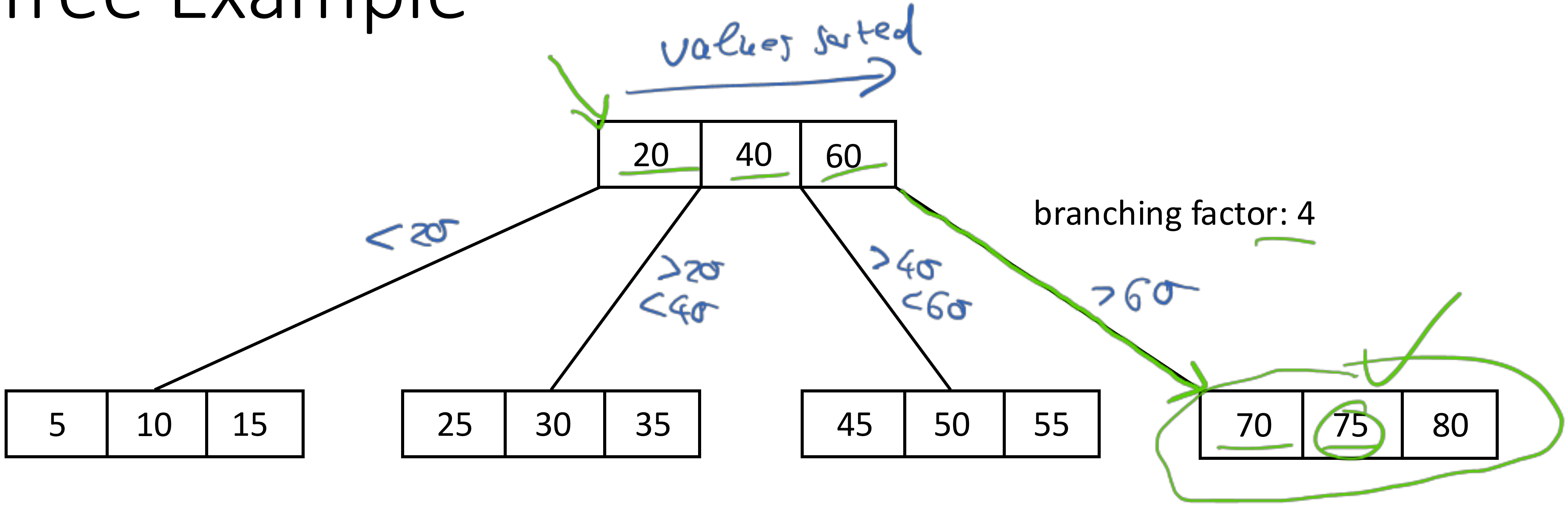
- Following a reference to a tree node requires memory access
- Memory access operations can be slow on mass storage devices

- Goal: Keep the number of memory access operations low

- B trees: make trees shallower (decrease # of levels)
  - Increase # of values stored in each node
  - Increases children of node

B Tree Example

Search: 75

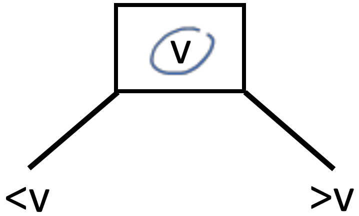


"balanced"

B Tree Properties

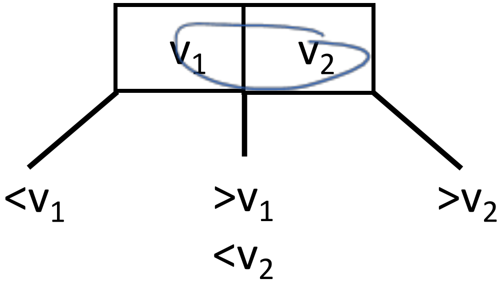
- Search trees
  - Values are ordered in node
  - Children are ordered
- Height H = # nodes on path from root to deepest leaf
- Self-balancing: Height grows in O(log N)
  - All leaves must be on the same level
  - All internal nodes must have (# of values)+1 children
  - Insertions can only happen into leaf nodes
  - B trees grow (add a new level) from the root upwards

B Tree Internal Node Types



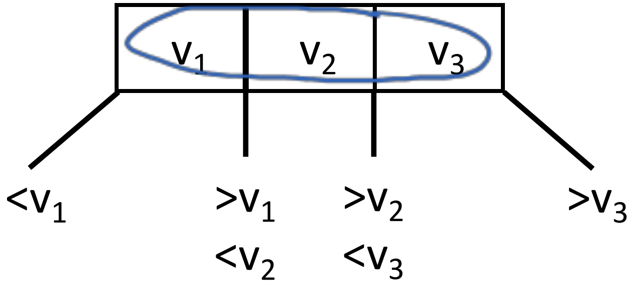
2-node

1 value, 2 children



3-node

2 values, 3 children

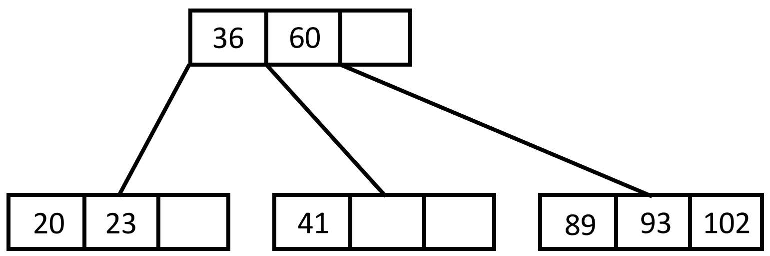


4-node

3 values, 4 children

2-3-4 Trees

2-3-4 Trees

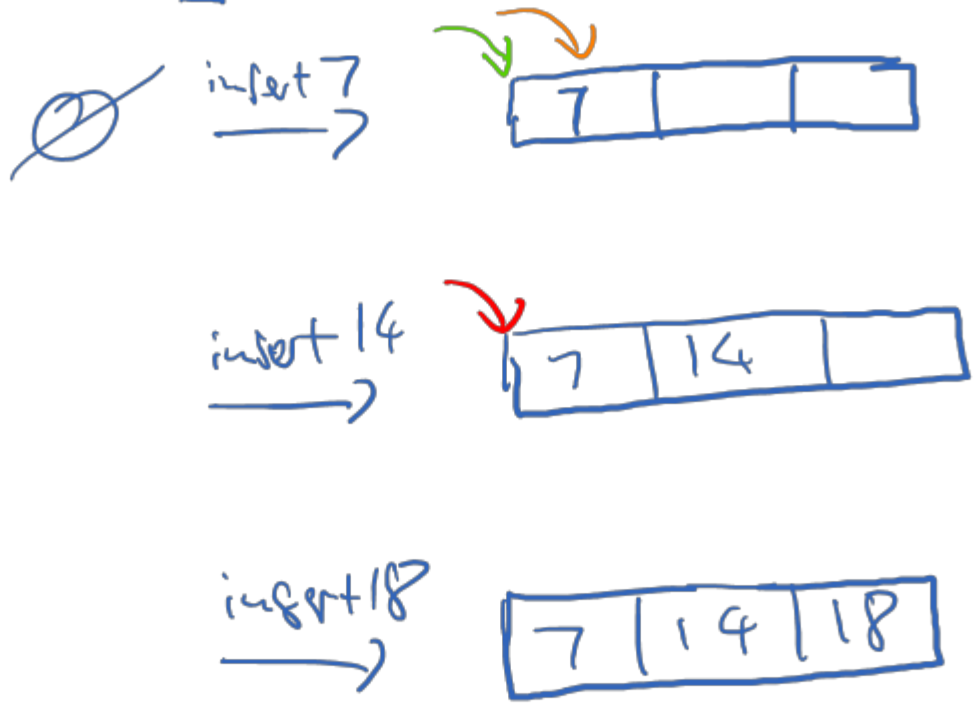


- B trees with 2-nodes, 3-nodes, and 4-nodes

- Insertion algorithm for new value:
  1. Perform search for value to find leaf to insert into
     - For every “full” node (with 3 values) on path to and including the leaf during search:
       - Split node (pre-emptive split)
       - Continue search at parent but don’t split it even if it is now full
  2. Insert new value into leaf at end of search

Insertion Example

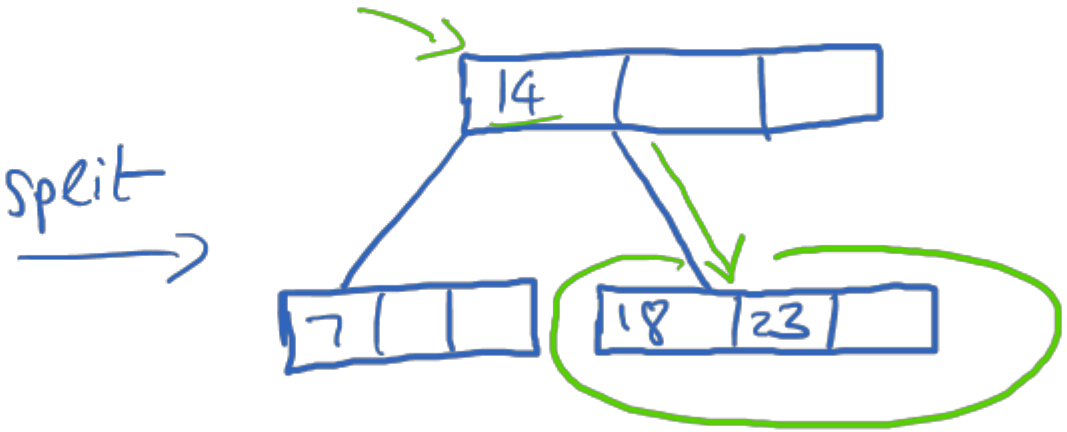
Insert 7, 14, 18 into empty tree.



Insertion Example

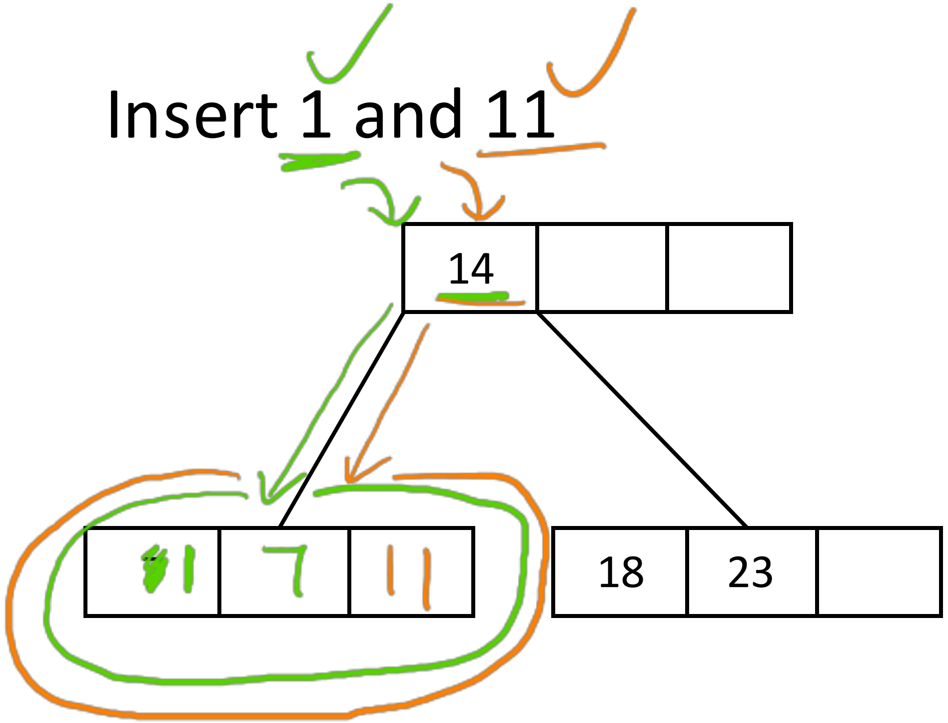
Insert 23

7	14	18



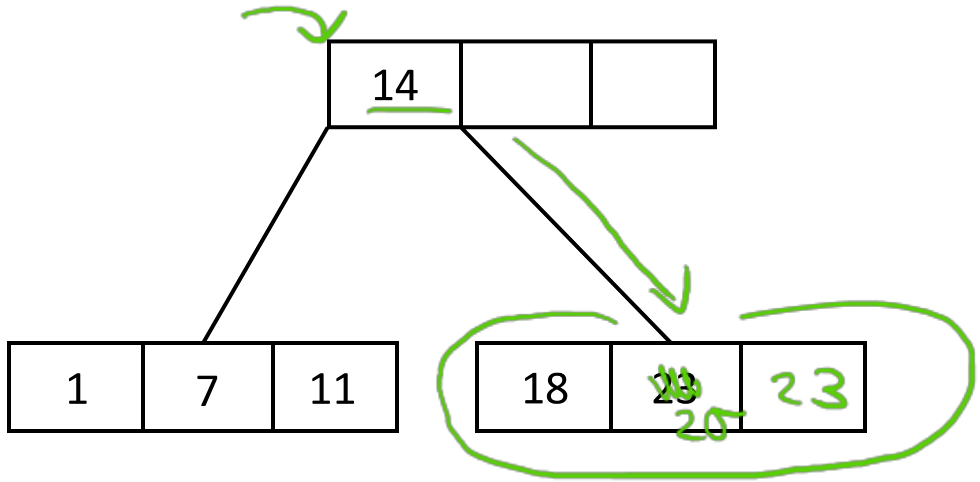
Insertion Example

Insert 1 and 11



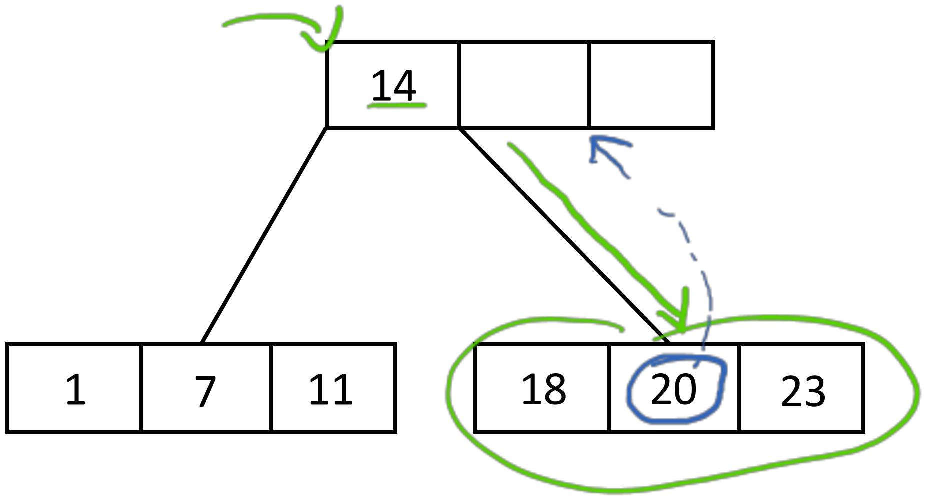
Insertion Example

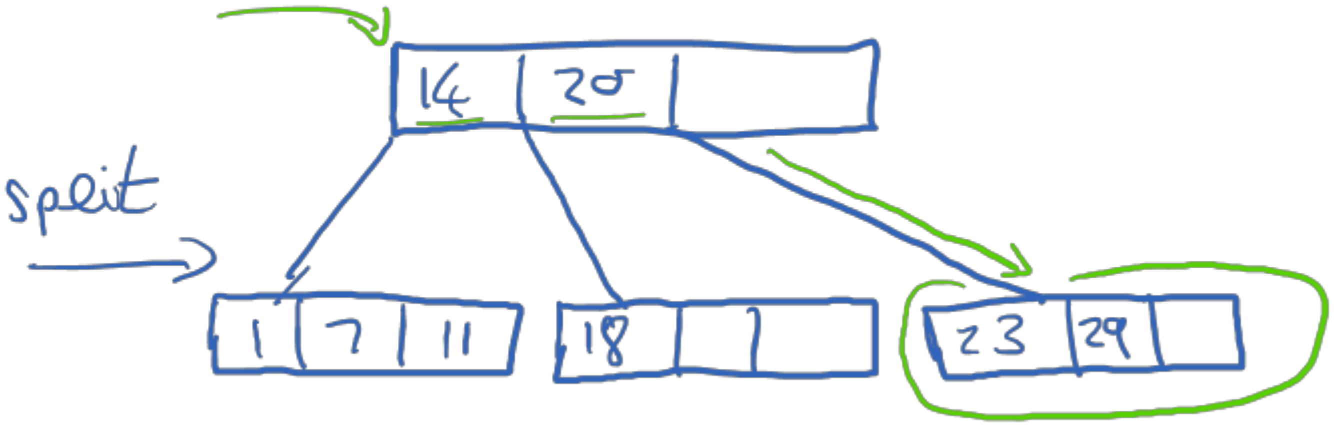
Insert 20



Insertion Example

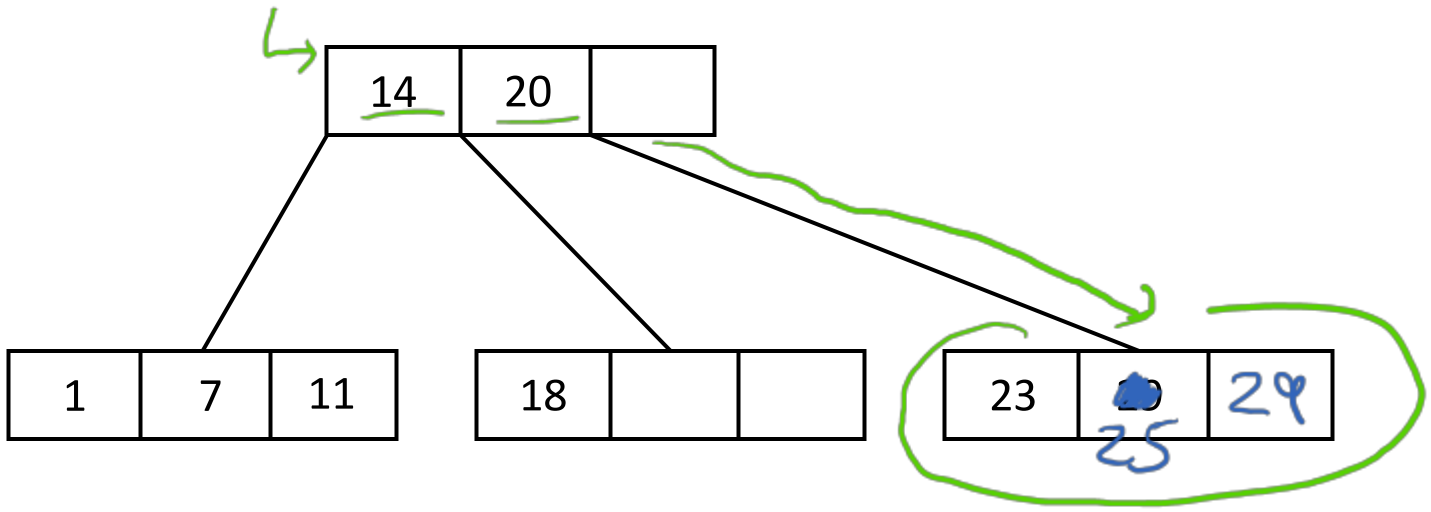
Insert 29





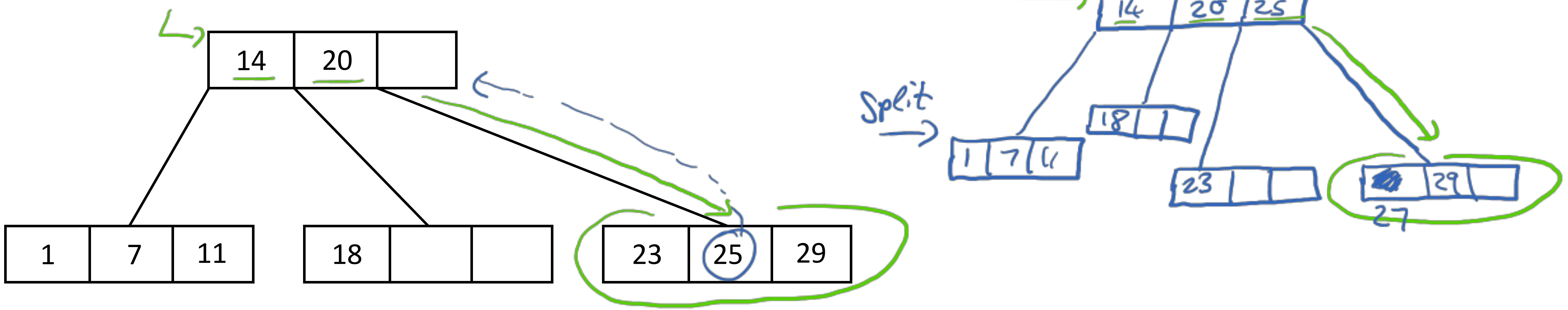
Insertion Example

Insert 25



Insertion Example

Insert 27



Insertion Example

Insert 10